GlobalLogic®
A Hitachi Group Company

Capstone Project

Hitachi Mobile Application

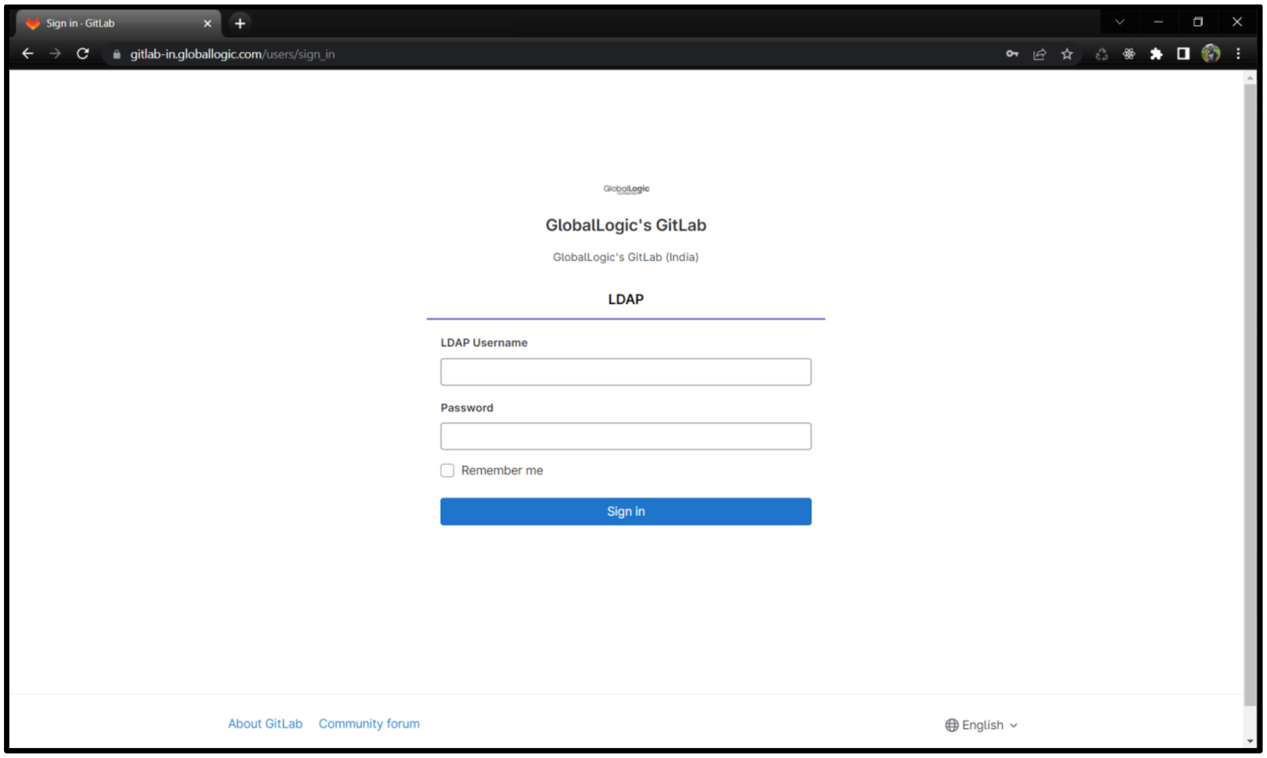
- Title of the Project - Hitachi Mobile Application
- Complexity - Medium
- Target Band - Band x / Band 1 / Band 2
- Skills: Java, JUnit Mockito, PostgreSQL.
- Time taken to complete: 3 Hours
- IDE: IntelliJ / Eclipse

Technology	Topics
Java	OOPs, JDBC, Abstract Class and Interfaces, Exception Handling, Collections And Generics, Streams API, Lambda Expression, JUnit5 And Mockito, PostgreSQL.0

Table of Contents

GITLAB GENERAL INSTRUCTIONS

1. Connect to VPN to access GlobalLogic GITLAB
2. Open the URL in the browser - https://gitlab-in.globallogic.com/users/sign_in

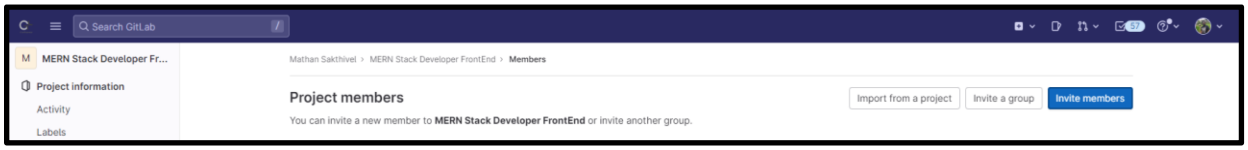


3. Login using GlobalLogic credentials.
4. Find a project to fork: Navigate to the project that you want to fork on GitLab. You can find projects by browsing the public repositories or by searching for a specific project.
5. Fork the project: Once you have found the project you want to fork, click the "Fork" button in the top right corner of the project page. This will create a copy of the project in your GitLab account.

6. Clone the forked project: Once you have forked the project, clone the project to your local machine using Git. To do this, open a terminal window and navigate to the directory where you want to clone the project. Then, run the following command:
   git clone [URL of the forked project]
7. Make changes to the project: Once you have cloned the project to your local machine, make any necessary changes to the project files using your preferred code editor.
8. Commit your changes: Once you have made changes to the project files, stage the changes and commit them to your local repository using the following commands:
   git add.
   git commit -m "Your commit message here"
9. Push your changes to GitLab: Once you have committed your changes to your local repository, push the changes to GitLab using the following command:
   git push origin [name of your branch]

10. The forked project will have the Private Access so explicitly add the users in the project members.

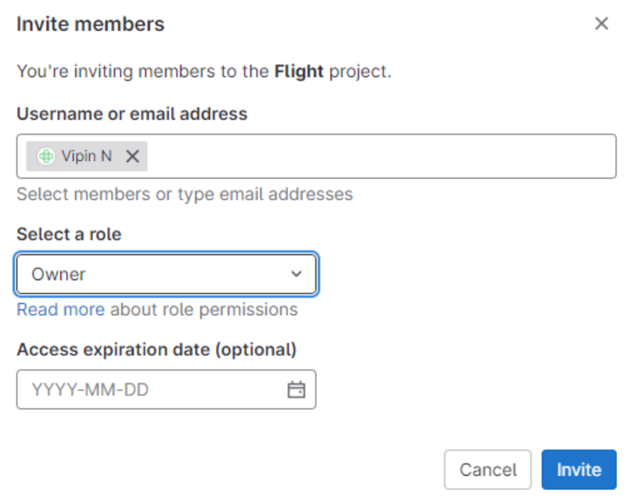
Click on Invite Members



Project members

You can invite a new member to **MERN Stack Developer FrontEnd** or invite another group.

Provide access to the Educators:
Username: Ex: minakshi.sharma@globallogic.com and vipin.n@globallogic.com
Role: Owner



Invite members

You're inviting members to the **Flight** project.

Username or email address

Vipin N

Select members or type email addresses

Select a role

Owner

Read more about role permissions

Access expiration date (optional)

YYYY-MM-DD

SYSTEM REQUIREMENTS

1. Eclipse/STS/IntelliJ IDEA
2. PostgreSQL
3. Java 17 or above

Hitachi Mobile Application

1. Problem Statement

Hitachi-Mobile is one of the world's top 10 telecom companies. Glo Logic has to create a Subscriber Identity Module (SIM)activation portal for automating the Subscriber Identity Module (SIM) activation process for the customers of Hitachi-Mobile.

Hitachi-Mobile customers can purchase a Subscriber Identity Module (SIM) by providing their first name, last name, email address, date of birth, etc., The customer will receive a Subscriber Identity Module (SIM)starter pack that contains information about the SIM.

By providing the Subscriber Identity Module (SIM)number that is included in the starter kit and the service number of the mobile number, customers can activate the Subscriber Identity Module (SIM)through this portal.

Additionally, the portal should support features like checking the SIM status, updating customer details, etc.

2. User Stories

User Story	Description	Acceptance Criteria
US01	As a customer, I want to have access to a feature that allows me to view a list of all my SIM card details.	● When I access this feature, it should display a list of all my SIM card details without requiring any additional input or technical knowledge. ● The list should include the following details for each SIM card: SIM ID, service number, SIM number, status, and unique number.
US02	As a customer, I want to be able to update the address associated with my account.	● When I access the feature, it should allow me to input my unique ID and the new address. ● The system should validate the unique ID to ensure it is associated with an existing account.

		● The system should validate the new address to ensure it is in a valid format. ● After I submit the new address, the system should update the address associated with my unique ID in the database.
US03	As an admin, I want to retrieve a list of all customers from the database. This feature should throw a `CustomerTableEmptyException` if there are no customers in the database.	● When I access the feature, it should display a list of all customers in the database without requiring any additional input. ● The list should include all relevant details for each customer. ● The information should be accurate and up-to-date, reflecting the current status of each customer. ● If there are no customers in the database, the feature should throw a `CustomerTableEmptyException`.
US04	As an admin, I want to fetch a list of SIM details with active status so that I can manage and monitor all the active SIM cards in the system.	● When I, as an admin, call the `fetchSIMDetailsWithActiveStatus()` method, it should return a list of `SIMDetails` objects that are currently active. ● If there are no active SIM cards in the system, the method should return an empty list. ● The method should handle any exceptions and not crash the system

		if there is an error fetching the SIM details.
US05	As an admin ,I want to retrieve the status of a SIM card, So that I can monitor and manage the SIM card's service.	● Given a valid SIM number and service number, when I call the getSimStatus function, then it should return the status of the SIM card. ● If I provide an invalid SIM number or service number, the system should throw a SIMDoesNotExistsException. ● The system should handle the exception and provide a meaningful error message to the user.

3. Table structure

i).Customer

Column Name	Description	Data Type	Not Null
customer_id	Unique Identification Number for customer	BIGINT	NOT NULL
date_of_birth	Date Of Birth of Customer	VARCHAR(20)	NOT NULL
email_address	Email Address of Customer	VARCHAR(20)	NOT NULL
first_name	FirstName of Customer	VARCHAR(20)	NOT NULL
last_name	Last Name of Customer	VARCHAR(20)	NOT NULL
id_type	Id type of customer like Adhar card or PAN Card or Vote ID details	VARCHAR(20)	NOT NULL
address	Address of Customer	VARCHAR(20)	NOT NULL
state	State Of Customer	VARCHAR(20)	NOT NULL
Primary Key(customer_id)			

ii). Sim_details

Column Name	Description	Data Type	Not Null
sim_id	Unique Sim Identification Number	INTEGER	NOT NULL
service_number	Service Number	BIGINT	NOT NULL
sim_number	Sim Number	BIGINT	NOT NULL
status	Status of Sim as “Active” or “InActive”	VARCHAR(20)	NOT NULL
customer_id	Unique Identification Number of Customer	BIGINT	NOT NULL
Primary Key - sim_id			
Foreign Key-customer_id - References Customer Table(customer_id)			

Relationship between above Tables /Entities

The relationship between the Customer and Sim_details entities is a One-to-Many relationship.

- Each customer can have multiple SIM cards. This is represented by the customer_id field in the `Sim_details` table, which is a foreign key referencing the `customer_id` in the `Customer` table.

- However, each SIM card can only be associated with one customer. This is represented by the `sim_id` field in the `Sim_details` table, which is unique for each SIM card.

So, one customer (customer_id`) can have many SIM cards (`sim_id`), but each SIM card is associated with only one customer. This is a classic One-to-Many relationship.

Note:- Create the above tables and add the sample data.

4.Implementation

Class / Method Description

This is a standalone Java application where a menu-driven interface needs to be developed to execute the following functionalities.

Note:- The above UserStories are converted into equivalent methods and the descriptions of the service methods provided below need to be implemented.

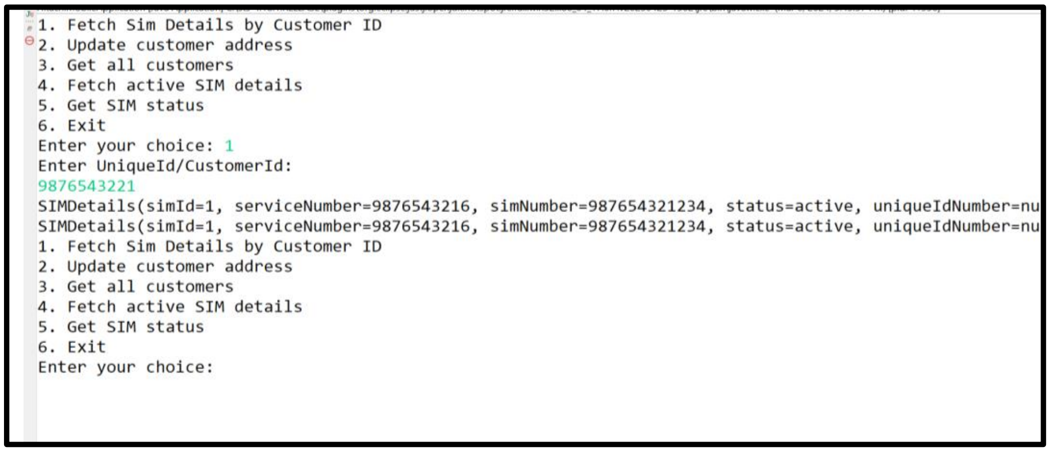
Entity Class	Service Methods	Description
Customer(customer_id ,dateOfBirth, emailAddress ,firstName ,lastName ,idType	fetchSIMDetails(unique_id_number)	The fetchSIMDetails(Long uniqueId) method is a public method that returns a list of SIMDetails objects. It takes a Long type parameter uniqueId. The purpose of this method is typically to retrieve the SIM details associated with

,address,state) Use Java17 Records		the provided unique ID.The `fetchSIMDetails(Long uniqueId)` method is a public method that returns a list of `SIMDetails` objects. It takes a `Long` type parameter `uniqueId`. The purpose of this method is typically to retrieve the SIM details associated with the provided unique ID. This method should handle exceptions properly. If the `uniqueId` related to whether the customer is not found or invalid, it should throw an appropriate exception, such as `IllegalArgumentException` or a custom exception like `CustomerDoesNotExistsException`. Note: unique_id_number is customer unique identification number or customer_id
	updateCustomerAddress(Long uniqueId, String newAddress)	The method `updateCustomerAddress(Long uniqueId, String newAddress)` is presumably a method that updates the address of a customer in a database or some other form of data storage. - `Long uniqueId`: This is likely the unique identifier for a customer. It's used to find the specific customer whose address needs to be updated. - `String newAddress`: This is the new address that will replace the customer's current address in the data storage. This method does not specify a return type, so it's assumed to be void. This means it doesn't return any value.
	getAllCustomers()	The `getAllCustomers()` method is a public method that returns a list of `Customer` objects. This method does not take any parameters. It is designed to retrieve all customers from a certain source, such as a database or a file.

		If the customer table or source is empty, it throws a \`CustomerTableEmptyException\`. This is a custom exception, likely defined elsewhere in your code, that is used to indicate that the operation cannot be completed because the customer table is empty.
SIMDetails(simId,serviceNumber,simNumber,status,customer_id)	fetchSIMDetailsWithActiveStatus() Write Lambda expression and use filter method to filter and retrieve the list of active simdetails	fetchSIMDetailsWithActiveStatus()- is a public method designed to fetch details of all active SIM cards in the system. It returns a list of SIMDetails objects, each representing an active SIM card.
	getSimStatus(long simNumber, long serviceNumber) Write Lambda Expression for above.	The \`getSimStatus(long simNumber, long serviceNumber)\` method is a public method that returns a \`String\`. This method takes two parameters: \`simNumber\` and \`serviceNumber\`, both of which are of type \`long\`. This method is designed to retrieve the status of a SIM card, given its SIM number and service number. The status is returned as a \`String\`. If the SIM card corresponding to the provided SIM number does not exist, it throws a \`SIMDoesNotExistsException\`. This is a custom exception, likely defined elsewhere in your code, that is used to indicate that the operation cannot be completed because the specified SIM card does not exist.

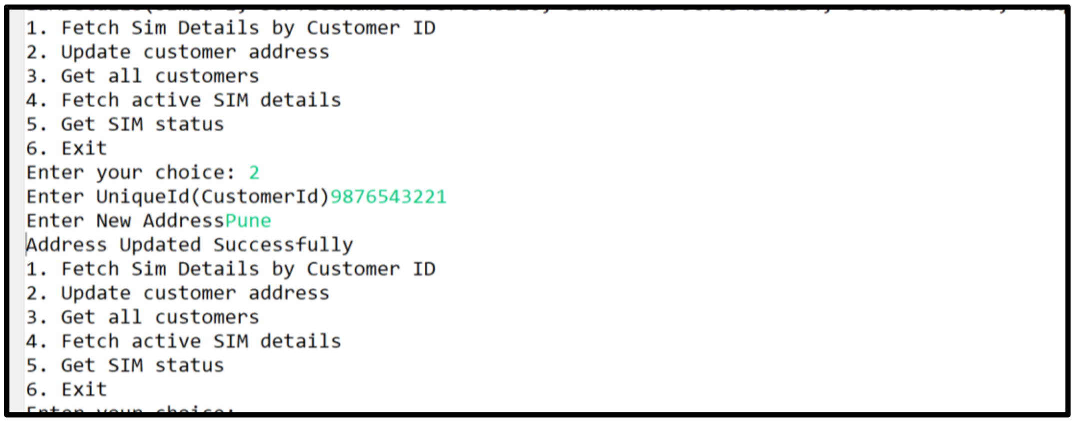
Output Screens

1. Fetch the Sim Details by Customer ID



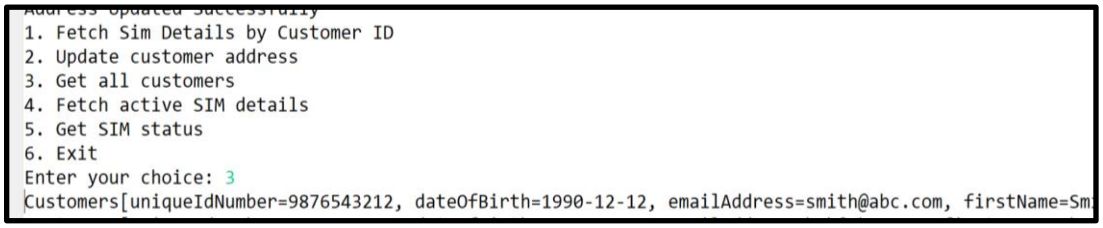
```
1. Fetch Sim Details by Customer ID
2. Update customer address
3. Get all customers
4. Fetch active SIM details
5. Get SIM status
6. Exit
Enter your choice: 1
Enter UniqueId/CustomerId:
9876543221
SIMDetails(simId=1, serviceNumber=9876543216, simNumber=987654321234, status=active, uniqueIdNumber=nu
SIMDetails(simId=1, serviceNumber=9876543216, simNumber=987654321234, status=active, uniqueIdNumber=nu
1. Fetch Sim Details by Customer ID
2. Update customer address
3. Get all customers
4. Fetch active SIM details
5. Get SIM status
6. Exit
Enter your choice:
```

2. Update Customer Address



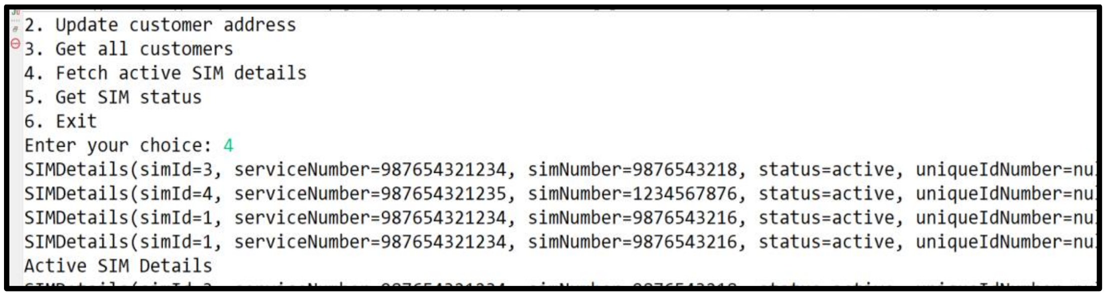
```
1. Fetch Sim Details by Customer ID
2. Update customer address
3. Get all customers
4. Fetch active SIM details
5. Get SIM status
6. Exit
Enter your choice: 2
Enter UniqueId(CustomerId)9876543221
Enter New AddressPune
Address Updated Successfully
1. Fetch Sim Details by Customer ID
2. Update customer address
3. Get all customers
4. Fetch active SIM details
5. Get SIM status
6. Exit
```

3. Get All Customer details



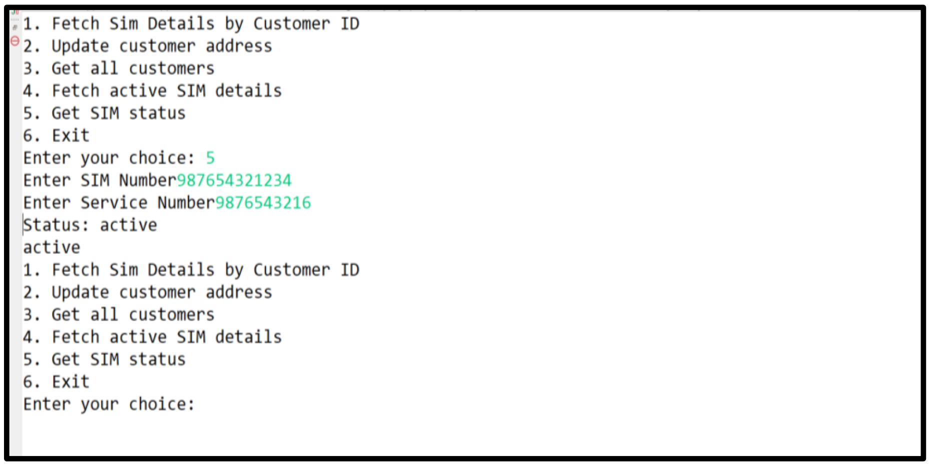
```
1. Fetch Sim Details by Customer ID
2. Update customer address
3. Get all customers
4. Fetch active SIM details
5. Get SIM status
6. Exit
Enter your choice: 3
Customers[uniqueIdNumber=9876543212, dateOfBirth=1990-12-12, emailAddress=smith@abc.com, firstName=Sm
```

4. Fetch Active SIM Details



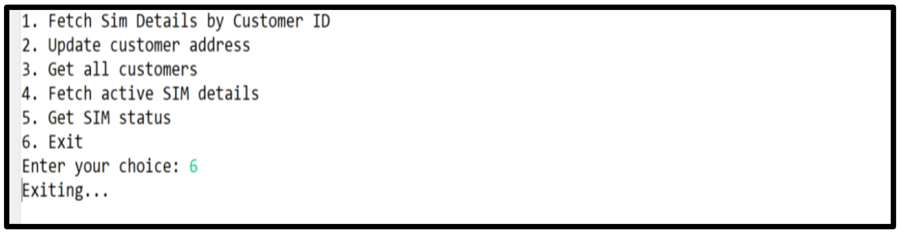
```
2. Update customer address
3. Get all customers
4. Fetch active SIM details
5. Get SIM status
6. Exit
Enter your choice: 4
SIMDetails(simId=3, serviceNumber=987654321234, simNumber=9876543218, status=active, uniqueIdNumber=nu
SIMDetails(simId=4, serviceNumber=987654321235, simNumber=1234567876, status=active, uniqueIdNumber=nu
SIMDetails(simId=1, serviceNumber=987654321234, simNumber=9876543216, status=active, uniqueIdNumber=nu
SIMDetails(simId=1, serviceNumber=987654321234, simNumber=9876543216, status=active, uniqueIdNumber=nu
Active SIM Details
```

5. Fetch Active SIM status



```
1. Fetch Sim Details by Customer ID
2. Update customer address
3. Get all customers
4. Fetch active SIM details
5. Get SIM status
6. Exit
Enter your choice: 5
Enter SIM Number987654321234
Enter Service Number9876543216
Status: active
active
1. Fetch Sim Details by Customer ID
2. Update customer address
3. Get all customers
4. Fetch active SIM details
5. Get SIM status
6. Exit
Enter your choice:
```

6. Exit



```
1. Fetch Sim Details by Customer ID
2. Update customer address
3. Get all customers
4. Fetch active SIM details
5. Get SIM status
6. Exit
Enter your choice: 6
Exiting...
```

5.Testing

- Create a test class named SIMDetailsServiceTest and write a JUni5t test for the fetchSIMDetailsWithActiveStatus() method in the SIMDetailsService class.

    - In the SIMServiceTest class
        - create a new method testFetchSIMDetailsWithActiveStatus(). This method will test the fetchSIMDetailsWithActiveStatus() method in the SIMDetailsService class.
        - Create a Mockito mock for the SIMService class. Use this mock to stub the fetchSIMDetailsWithActiveStatus() method to return a list of SIMDetails objects with active status.
        - Call the fetchSIMDetailsWithActiveStatus() method on the SIMService mock.
        - Verify that the returned list is not null, not empty, and that all SIMDetails objects in the list have active status.

Instructions to Create and Execute the Project

- Please download the code structure from the provided GitLab link. The project structure can be found under the 'com.gl.app' package and its subpackages.
- Please design the necessary database schema to store Customer and SIMDetails.
- Create a Menu driven Java Project for above given methods
- Use PostgreSQL as the database management system.
- In addition to implementing the functionalities mentioned above(i.e method implementation)
    - Read Input from the keyboard using Scanner class.
    - You should include code for establishing a database connection, executing queries, and handling exceptions.
    - Use Java 8 Lambda Expressions,Streams API.
    - Field Validations(Acceptance Criteria)
        - Validate the Customer details(name,email,sim_number).
        - sim_id should have a unique value in the sim_details table.
        - Customer_id should have unique values in the customer table.
        - Establish the above-mentioned relationship between the tables to ensure the correctness of the data.
        - Handle and display errors gracefully to the user in case of invalid inputs or other issues.

Evaluation Metrics

Sl. No.	Assessing Parameter	Marks

1	User Stories 5* 7 marks	35 marks
2	Usage of Streams and Records	5 marks
3	Testing Module	5
4	Exception Handling and following best practices	5
	Total Marks	50 Marks